Data Encapsulation

# Before Class

1. Familiarise yourself with the data encapsulation concept. Explain why it is necessary to hide the implementation details of a class.

Data encapsulation to pojęcie, które oznacza ukrywanie szczegółów implementacyjnych klasy i dostęp do jej danych tylko za pośrednictwem zdefiniowanych publicznie metod. Jest to jeden z filarów programowania obiektowego.

Bezpieczeństwo i Ochrona Danych: Ukrywanie szczegółów implementacyjnych pomaga w zabezpieczaniu danych przed niepożądanym dostępem i modyfikacją. Dostęp do danych może być kontrolowany poprzez ustawianie odpowiednich metod dostępu (gettery i settery), co pozwala na lepszą kontrolę nad danymi.

Uniezależnienie Implementacji: Ukrywanie szczegółów implementacyjnych pozwala na zmianę wewnętrznej struktury klasy bez wpływania na zewnętrzny kod korzystający z klasy. Dzięki temu możliwa jest poprawa i rozwinięcie implementacji bez konieczności zmiany kodu korzystającego z klasy.

Ułatwia Zrozumienie i Używanie: Ukrywanie implementacji sprawia, że klasy stają się bardziej intuicyjne w użyciu, ponieważ programista korzystający z klasy nie musi znać wszystkich szczegółów jej wewnętrznej struktury. To pozwala na bardziej czytelny i zrozumiały kod.

Zmniejsza Zależności: Ukrywanie implementacji zmniejsza zależności między różnymi częściami programu. Kod, który korzysta z klasy, może być bardziej niezależny od jej wewnętrznej struktury, co ułatwia późniejsze modyfikacje i utrzymanie systemu.

Zachowanie Interfejsu: Ukrywanie szczegółów implementacyjnych pozwala na utrzymanie spójnego interfejsu klasy, nawet jeśli wewnętrzna implementacja ulegnie zmianie. To zapewnia stabilność dla kodu korzystającego z klasy.

1. Explain the meaning of access modifiers: public, private and protected.

Public (publiczny):

Znaczenie: Publiczne elementy są widoczne i dostępne z dowolnego miejsca w programie.

Przykład: Jeśli zmienna lub metoda jest oznaczona jako public, oznacza to, że można się do niej odwołać z dowolnej części programu.

Private (prywatny):

Znaczenie: Prywatne elementy są widoczne tylko w obrębie tej samej klasy, w której są zdefiniowane. Nie są dostępne z zewnątrz klasy.

Przykład: Jeśli zmienna lub metoda jest oznaczona jako private, to tylko metody w tej samej klasie mają do niej dostęp.

Protected (chroniony):

Znaczenie: Chronione elementy są widoczne w obrębie tej samej klasy oraz w klasach dziedziczących (podklasach).

Przykład: Jeśli zmienna lub metoda jest oznaczona jako protected, to jest dostępna dla klasy, w której jest zdefiniowana, oraz dla jej klas dziedziczących.

W skrócie:

public: Dostępne wszędzie.

private: Dostępne tylko w obrębie tej samej klasy.

protected: Dostępne w obrębie tej samej klasy i dla klas dziedziczących.

1. Watch the video “Java encapsulation”:

<https://youtu.be/eboNNUADeIc>

1. What are getters and setters methods and how they support data protection.

Gettery (metody pobierające):

Funkcja: Zwracają wartość prywatnego pola.

Sposób użycia: Gettery są często nazwane w taki sposób, że zaczynają się od słowa "get", a następnie podają nazwę pola, którego dotyczą.

Jeśli mamy prywatne pole imie w klasie Osoba, getter może nazywać się getImie(), a jego zadaniem będzie zwracanie wartości pola imie.

Settery (metody ustawiające):

Funkcja: Ustawiają wartość prywatnego pola.

Sposób użycia: Settery są często nazwane w taki sposób, że zaczynają się od słowa "set", a następnie podają nazwę pola, którego dotyczą. Oprócz tego, przyjmują argument, który jest nową wartością do ustawienia.

Jeśli mamy prywatne pole imie w klasie Osoba, setter może nazywać się setImie(String noweImie), a jego zadaniem będzie ustawianie wartości pola imie na noweImie.

Jak gettery i settery wspierają ochronę danych:

Kontrolowany dostęp: Pole jest prywatne, co oznacza, że nie można bezpośrednio odwołać się do niego z zewnątrz klasy. Gettery pozwalają na kontrolowane odczytywanie wartości, a settery na kontrolowane ustawianie nowych wartości.

Walidacja danych: Przy użyciu setterów można dodawać logikę walidacyjną, aby sprawdzić poprawność nowych wartości przed ich ustawieniem. Na przykład, możemy sprawdzić, czy nowa wartość liczby jest dodatnia przed jej ustawieniem.

Modyfikacja wewnętrznej implementacji: Jeśli zdecydujemy się w przyszłości zmienić sposób przechowywania danych (np. zmienna lokalna na obiekt), możemy to zrobić bez wpływu na kod korzystający z getterów i setterów, ponieważ interfejs (metody publiczne) pozostaje niezmieniony.

1. Familiarise yourself with entering data by using keyboard. From the course textbook, read the Chapter 3 (Input and Output).

# During Class

1. Watch the video ‘What is Encapsulation’

<https://youtu.be/bSpPwVFEbO8?feature=shared>

1. Give examples of three real-world objects where encapsulation is used.

## Access modifiers

1. Define the DrivingLicense class, containing the following attributes: driver's name and surname, address, postal code, city, driving license number, year of issue and driving license category. Use private access modifiers when declaring attributes. Then, create a driving license and try to assign values to the attributes. What message are you getting? After that, change the access modifiers from private to public and try to assign values once again. What can you observe?

public class DrivingLicence {  
 private String name;  
 private String surname;  
 // rest of fields  
 // ...  
}

public class DrivingLicenceTest {  
 public static void main(String[] args) {  
 DrivingLicence dl = new DrivingLicence();  
 dl.name = "Jack";  
 // ...  
 }  
}

1. Complete the DrivingLicense class. Define the display() method that displays the driving license (all data contained on the driving license). Try to display the data in an attractive form. What access modifier will you apply to the defined method?

## Getters and Setters

1. Apply the private access modifier to all attributes of the DriverLicense class. Then, create get and set methods for each of the attributes. After that, modify the program and the method displaying the driving license. Replace attributes names with get and set methods.
2. Add a toString() method in the DrivingLicense class to return driving license information. Use getter methods to get the driving license data. Then, check the method in action.
3. In the DrivingLicense class, modify the setter method for the year of issue attribute. The valid value of the attribute should be greater than or equal to 1980 and less than or equal to the current year. If the given value is different, do not change the attribute.
4. Modify the setName() method in the DrivingLicense. Regardless of the given name value, the value of the attribute should begin with a capital letter. Replace the remaining letters with lowercase. Then check the method in action. Tip. Use the methods available in the String class.

# After Class

1. The class String includes methods for examining individual characters of the sequence, for comparing strings, for searching strings, for extracting substrings, and for creating a copy of a string with all characters translated to uppercase or to lowercase. You can use the methods in your programs. Write a program that, for the text:

"Have a nice day!"

calculates and displays:

* 1. The number of characters
  2. The first 9 characters
  3. True if the text ends with the suffix "day!"
  4. True if the text is not empty
  5. Index of the last occurrence of the character ‘e’
  6. The text in which each space character has been replaced with a minus sign
  7. The text converted to upper case

1. The Product class describes food products using two attributes: the product name and whether the product is vegetarian. Define the class, its attributes, and getter and setter methods for all attributes. Then, create a product, set attributes' values and display product information.
2. Choose any object. Then define a class that describes such objects. Hide data about an object using data encapsulation.
3. Define a class Person with two attributes describing a person: name (String) and age (int). Apply data encapsulation. Define a constructor with the parameters name and age to assign initial values of object’s attributes. Define access and modification methods for each attribute (getter and setter methods). Use method names according to the naming convention. Then define a method isAdult() that returns true if a person is an adult (person has at least 18 years) or false otherwise. Finally, define a method that returns a string representation of the object (name and age, separated by comma). Sample result:

Person p = new Person("Anna",21)  
p.getAge() returns 21  
p.isAdult() returns true  
p.setAge(17)  
p.isAdult() returns false  
p.toString() returns "Anna,17"

1. Define a class Counter that allows you to create a counter of integer type. The initial value of the counter is 0. The class includes an increase() method that increases the value of the counter by 1 and a decrease() method that decreases the value of the counter by 1. Also create the overloaded methods increase(int n) and decrease(int n) that allow you to increase or decrease the value of the counter by the value of n. Add a value() method in the class that returns a counter value. Sample result:

Counter c = new Counter()  
c.value() returns 0  
c.increase()  
c.increase()  
c.decrease()  
c.increase(5)  
c.decrease(2)  
c.value() returns 4

1. Define a Point class that contains two attributes: x and y, of integer type (int), describing the coordinates of a point on the plane. The class constructor contains two parameters and allows you to initialize the object's attributes. Create an isX() method in the class that returns true if the point is on the x-axis and false otherwise. Create an isY() method in the class that returns true if the point is on the y-axis or false otherwise. Add a method in the class to represent the object as text that returns the coordinates of the point in the format "P(x,y)". Sample result:

Point(3,0)  
isX() returns false  
isY() returns true  
toString() returns "P(3,0)"

1. Define a Book class with a **title** attribute of string type and a **pages** attribute of integer type. Apply data encapsulation. Create an accessor and modifier for each attribute. In the method that modifies the number of book pages, change the value of the object attribute only when the specified number of book pages is positive. Sample result:

Book b = new Book()  
b.setPages(3)  
b.setPages(-4)

1. Define a MyArrays class that contains two static methods: odd(int[] array), which returns the number of positive odd values in the array, and above(int[] array), which returns the sum of the numbers in the array that are greater than the arithmetic mean of the array elements. Sample result:

arr1 = {3,2,-5,4,1,-7}  
arr2 = {5,2,7,4,2}  
MyArrays.odd(arr1) returns 2  
MyArrays.above(arr2) returns 12

1. Define a Clock class that allows you to create clocks. The class has two integer (int) attributes: hour and minute. The hour attribute can take values from 0 to 23, and the minute attribute can take values from 0 to 59. The constructor of this class, containing the parameters (int hour, int minute), allows you to initialize the clock with the given values of hours and minutes. The class also has an addMinute() method that moves the clock forward one minute. Apply data encapsulation. For the hour and minute attributes, define access and modification methods.